This is The Way: Vision-Based End-to-End Planning for Autonomous Driving

Anisha Palaparthi
Stanford University
anishapv@stanford.edu

Arpit Dwivedi
Stanford University
dwivedi7@stanford.edu

Purushotham Mani
Stanford University
purush@stanford.edu

Abstract

End-to-end autonomous driving has attracted substantial interest in recent years. In particular, our work focuses on predicting the vehicle's future trajectory, based on the past dynamics information, visual input of the current timestep from onboard cameras, and a specified driving intent. Much of the existing work focuses on leveraging modular deep neural networks to build end-to-end architectures on typical driving scenarios. However, it remains an open question on how to address the end-to-end driving problem in uncommon or unexpected situations in which the vehicle must adapt to its surrounding conditions quickly and without harming its environment. In this work, we aim design a model architecture that explores this problem on the Waymo Driving Dataset. Our architecture takes advantage of transformer-based vision models, Long Short-Term Memory based encoders, and insights from from language models to achieve a significant performance increase over the baseline in a low data regime. Our results suggest that this architecture not only allows the model to generalize well to unseen data, but also has the potential to realize even better performances with more compute resources and data.

1. Introduction

Autonomous driving technology has seen rapid advancement over the past decade. The prevailing approach in the industry relies on a modular architecture, dividing the driving pipeline into distinct components such as perception, localization/mapping, prediction, planning, and control. This decomposition offers a practical balance between performance, safety, and explainability, enabling commercial success in Level 2 (partial automation) to Level 3 (periodically autonomous) driver assistance systems.

However, despite progress in lower levels of autonomy, the widespread deployment of higher-level (L4+, with limited or no human control) autonomous systems remains limited. Several challenges contribute to this gap. First, the complexity of the modular pipeline—often comprising twenty or more interconnected modules—introduces significant overhead. Each module operates with limited compute resources, and the large number of interfaces between them hampers seamless information flow and joint optimization. Potential conflicts between local and global objectives are an additional concern.

Second, traditional architectures often suffer from poor generalization. Handcrafted heuristics and narrowly trained models struggle to handle rare and unexpected driving scenarios, what we call "*long-tail scenarios*". Over time, the accumulation of brittle, case-specific solutions can make the system increasingly difficult to maintain and scale.

Third, these systems are typically tightly coupled to specific software and hardware platforms, making it difficult to adapt them across different vehicle models/platforms.

In light of these limitations, we explore an *end-to-end approach to autonomous driving* that simplifies the architecture while aiming to enhance generalization and adaptability. Specifically, our system predicts the vehicle's future trajectory over a 5-second horizon, based on:

- the vehicle's past 4-second trajectory sampled at 4 Hz (a $(16, 6)$ sequence of $(x, y, v_x, v_y, a_x, a_y)$),
- 3 front-facing camera images at the current timestep
- a high-level driving intent
  ("go left", "go straight", "go right", or "unknown").

The model outputs a predicted trajectory over the next 5 seconds at 4 Hz, represented as a $(20, 2)$ sequence of future x-y positions. Furthermore, given the size of the data records (each record contains up 3 GB of data), we will develop our model under a low-data regime; this further motivates the problem of designing such an end-to-end approach without as much memory and computational resources.

This end-to-end formulation offers several advantages:

- **Simplified Architecture:** The entire decision-making pipeline is encapsulated in a single deep neural network module, reducing system complexity and increasing integration. This unified structure enables end-to-end optimization, avoiding the challenges of reconciling potentially conflicting local objectives across modules. As a result, training becomes more streamlined and globally coherent, since all components are optimized together under a single loss function.

- **Improved Generalization:** By incorporating multi-modal learning and leveraging the reasoning capabilities of language models and transformer-based vision models, the system is better equipped to handle the long-tail scenarios.

- **Enhanced Adaptability:** Since our model outputs predicted trajectories rather than direct control commands, it can be more easily deployed across different vehicles—each of which can use its own low-level controller to track the predicted path.

2. Related Work

Early deep models used recurrent networks to model motion. For example, Zhu et al. (2019) introduced an Attentive Recurrent Neural Process (ARNP) [18], which combines RNNs with neural processes to predict a probability distribution over a vehicle's future path given its past trajectory. Similarly, Kim et al. (2021) developed a driving-style–conditioned CVAE [6]: a DeepConvLSTM first classifies the driver's style from in-vehicle sensor data, and then a CVAE generates diverse future trajectories conditioned on that style and the recent motion history. These models capture agent dynamics and multi-modal futures via learned sequence models.

More recently, transformer and attention models have dominated motion forecasting by treating trajectories like sequences. For instance, Waymo's MotionLM [13] discretizes agent trajectories into tokens and applies a standard language-modeling objective. It autoregressively generates joint, multi-agent future sequences in a single pass, achieving state-of-the-art results on the Waymo Open Motion Dataset. Such transformer-based methods excel at modeling long-range interactions and multi-modal futures, though they typically require large training sets and pre-defined motion tokens. Notably, MotionLM focuses on agent dynamics and interactions, not on raw sensory images.

Another line of work explicitly incorporates structured map context. VectorNet [3] (Gao et al., 2020) represents both agents and map elements as polylines in a hierarchical graph, avoiding the lossy rasterization of images. By operating directly on vectorized HD maps and trajectories, it achieves competitive or better performance than CNN-based approaches on benchmarks like Argoverse. Likewise, MultiPath++ [15] (Varadarajan et al., 2022) encodes road lane geometries and agent states as compact polylines and fuses them via a multi-context gating mechanism. This context-aware fusion of map and motion information yields state-of-the-art accuracy on both the Argoverse and Waymo motion challenges. These vectorized models leverage rich, interpretable map features to improve prediction.

In contrast, end-to-end vision-based methods predict trajectories directly from raw images, learning scene understanding implicitly. ChauffeurNet [1], for instance, predicts ego-vehicle trajectories using surround-view cameras and a high-level navigational command (e.g., "turn left"). However, its pipeline relies on a pre-processed top-down semantic representation that includes rendered road layouts, objects, and traffic signals—making it only partially end-to-end. Our setup shares the same prediction objective but removes this intermediate perception step, working directly from RGB images, motion history, and natural language intent. Advances in transformer-based vision backbones have made this design feasible: Swin Transformer [8], a hierarchical ViT pretrained on ImageNet, has shown strong performance on diverse vision tasks. We adopt Swin to extract spatial and semantic features from front-facing camera views, enabling our model to leverage fine-grained scene cues such as lane markings and dynamic actors without relying on external maps or perception modules.

A more novel line of work incorporates textual or intent information into prediction. VisionTrap [12] adds textual scene descriptions (generated by vision-language models) as supervisory signals to guide a trajectory predictor. In their model, "textual descriptions generated by a VLM and refined by an LLM" help teach the model which semantic cues to focus on. Another approach, Trajectory-LLM [17], goes further by treating language as input: it feeds short text descriptions of multi-agent interactions (e.g. "cars at stop sign turning right") into an LLM, which then produces corresponding trajectories. In effect, Trajectory-LLM uses language models to synthesize realistic training trajectories from semantic prompts. These works show that high-level semantics (whether generated or hand-crafted) can guide motion prediction. Our method also leverages language, but in a simpler way: we embed a small set of natural-language intents ("go straight", "turn left", etc.) via a pretrained MiniLM, producing a dense intent vector. To our knowledge, few prior works have used natural language to encode driver intent in this manner.

In summary, prior trajectory prediction models can be broadly grouped into several categories. RNN and VAE-based approaches [18, 6] effectively model agent dynamics and interactions, while transformer-based models [13] offer improved capacity for handling complex, multi-agent scenarios—though they often overlook visual semantics. Map- and image-based methods [15, 3] incorporate environmental context through structured vector inputs or raw pixels, enabling stronger scene understanding. Vision-based approaches, such as ChauffeurNet [1], leverage raw scene imagery and privileged representations to predict motion plans, and are notable for reducing the need for extensive real-world data through effective use of imitation learning and synthetic supervision. More recently, language-augmented models [12] have begun to bridge vision and semantics, introducing high-level reasoning and improving interpretability, though this area remains relatively under-explored. Compared to these, our approach uniquely fuses a vision backbone (Swin) with RNN-encoded dynamics and a language-model-based intent embedding.

In our view, the cleverest recent advances include models that fuse multiple contexts (anchors + maps in MultiPath++[15], token-based sequence modeling in MotionLM[13]) and vision-language models like VisionTrap that exploit textual semantics [12].

Overall, the field has rapidly shifted to deep learning solutions; virtually all top approaches in recent challenges (e.g. Waymo, Argoverse) are learned, with minimal "by-hand" heuristics left, reflecting the power of these modern methods.

3. Dataset and Features

The dataset used in this project is a curated subset of the Waymo Open Dataset [9], specifically selected for the task of intent-aware vehicle trajectory forecasting. Given the substantial scale of the full dataset—which exceeds the computational resources available for this project—we extracted a manageable 3GB subset, consisting of 1000 training samples, 100 validation samples, and 263 test samples. Each sample includes:

- Three RGB images from the front-facing cameras (front, front-left, and front-right), capturing the forward driving context relevant for trajectory prediction.
- 4 seconds of past vehicle dynamics, sampled at 4 Hz (16 steps), with each timestep providing six features: position (x, y), velocity (v_x, v_y), and acceleration (a_x, a_y).

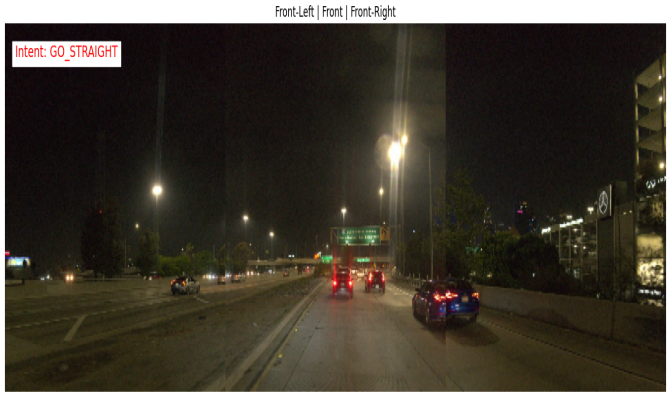

Figure 1: Example input showing the three front-facing camera views (front, front-left, front-right) along with the associated driving intent annotation.

- A high-level driving intent, represented as a one-hot encoded vector over four classes: "go straight", "turn left", "turn right", and "unknown".
- A target output consisting of 20 future trajectory waypoints over the next 5 seconds, capturing the ego vehicle's motion in 2D space (x, y).

Due to computational limitations, the model uses only the three front-facing camera views, excluding side and rear perspectives. An example input image is shown in Figure 1.

Several preprocessing steps were applied to prepare the data for modeling. The raw images were first converted into tensors and restructured into the [batch size, cameras, channels, height, width] format to support multi-view input from front-facing cameras. The intent labels, originally stored as integer class indices (ranging from 0 to 3), were converted into one-hot encoded vectors to facilitate compatibility with neural network architectures. For models leveraging pretrained vision encoders—we applied the standardized image preprocessing pipeline used during encoder's training. This included resizing images while maintaining aspect ratio, performing a center crop to 224×224 pixels, and normalizing pixel values using CLIP's mean and standard deviation statistics. These transformations ensure the input distribution matches the expectations of the pretrained model, enabling more effective feature extraction.

To further enhance generalization, we implemented a data augmentation pipeline with commonly used visual perturbations—random resized crops and color jitter. This augmentation strategy improves robustness to visual variations. Altogether, this comprehensive preprocessing strategy ensured the dataset was well-conditioned for our multimodal deep learning task that integrate visual, temporal, and semantic inputs to accurately predict future vehicle trajectories.

4. Methodology

We have designed a model architecture that takes as input the past trajectory dynamics, 3 images from the vehicle's cameras at the end of this trajectory, and an intent: to go straight, go left, or go right (or unknown). The model then predicts the future trajectory of the vehicle's position as output.

More specifically, we can define the past trajectory dynamics as $\mathbf{T} \in \mathbb{R}^{16\times 6}$. There are 16 previous timesteps in the data, and for each timestep where are given the x and y position, velocity, and acceleration of the vehicle. We can define the three RGB images, each of 224x224 pixels, as $\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3} \in \mathbb{R}^{224\times 224\times 3}$. Finally, we can define the intent as $\mathbf{I} \in \{0, 1, 2, 3\}$. The summary of inputs to the model is therefore given by:

$$\mathbf{T} \in \mathbb{R}^{16\times 6}, \mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3} \in \mathbb{R}^{224\times 224\times 3}, \mathbf{I} \in \{0, 1, 2, 3\}$$

The output is 20 timesteps of the future trajectory's position, namely $\mathbf{T}_{future} \in \mathbb{R}^{20\times 2}$.

The model architecture will comprise of 4 modules: (1) vision backbone, (2) dynamics encoder, (3) intent embedding and encoder, and (4) a final fusion layer. The vision backbone module takes $\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}$ as input and outputs an image feature vector $\mathbf{F}_{\text{img}} \in \mathbb{R}^D$. The dynamics module takes as input $\mathbf{T}$ and outputs an dynamics encoding $\mathbf{F}_{\text{dyn}} \in \mathbb{R}^L$. The intent module takes as input $\mathbf{I}$ and outputs an intent encoding $\mathbf{F}_{\text{int}} \in \mathbb{R}^E$. The final fusion layer takes three outputs described above concatenated together as input $\mathbf{F} = [\mathbf{F}_{\text{img}}\mathbf{F}_{\text{dyn}}\mathbf{F}_{\text{int}}] \in \mathbb{R}^{D+L+E}$ and outputs the predicted future trajectory $\mathbf{T}_{future} \in \mathbb{R}^{20\times 2}$. Notice that D, L, E are hyperparameters of the model architecture and are chosen to be $D = 256, L = 64, E = 16$.

Figure 2 illustrates this model architecture, which has a model capacity of 49.5 M parameters in total. To develop this model, we incrementally built up the various components and evaluated performance when trained on a subset of the dataset. These 20 versions of the model architecture and their experimental results are documented in Table 1. Below, we will discuss in further depth each of these components of the final architecture.

4.1. Vision Backbone

The first module of the model architecture is a vision backbone that provides encodes the 3 camera images from the front-left, front, and front-right of the vehicle into a D-dimension vector. Noting the high performance of transformer-based vision models in recent years, we opt to use a vision transformer for the purposes of the vision backbone. However, given the small dataset we are working with, we use a pre-trained vision transformer model (pre-trained on ImageNet [2]) from which we remove the classification head, and we finetune the model on our driving dataset by freezing the weights of the transformer model except for the final 2 transformer blocks and the normalization layers. Finally, we concatenate the R-dimensional image features (where R is the transformer output dimension) for the 3 input images extracted by the vision transformer and apply a feedforward layer to encode the 3R-dimensional image features into a D-dimension image encoding.

For the vision transformer, we considered 2 such variants pre-trained on ImageNet: a tradition tiny ViT with a patch size of 16 (approximately 9.7M parameters) and a small Swin Transformer with a patch size of 4 and window size of 7 (with approximately 49.5M parameters). While the ViT provided faster training and evaluation, it focuses more on global understanding of images. The Swin Transformer, on the other hand, is more equipped to extract low-level feature relationships, which is particularly relevant for our task since the images contain important low-level features such as stop signs, lane markings, etc.

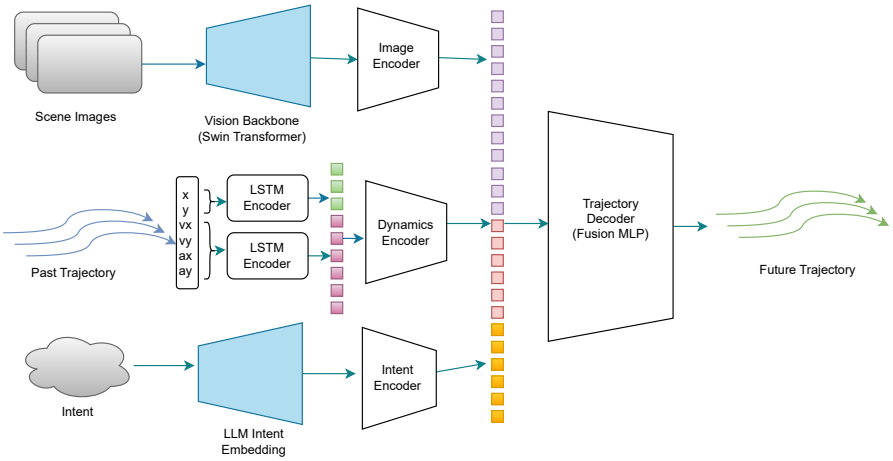

Figure 2: Model Architecture

4.2. Dynamics Encoder

To encode the dynamics of the past trajectory data (which, we can recall, includes x,y position, velocity, and acceleration for the last 16 timesteps), we first split these features into two separate vectors, one containing the positional information and other containing the kinematics information of velocity and acceleration. We then train two LSTM models as the positional and kinematics encoders, each of which provides final hidden state (an L-dimension vector) that encodes the past trajectories. The LSTM's have a hyperparamter hidden state size, which we choose to be 32. Finally, we have a final 2-layer MLP layer that fuses the LSTM outputs into a single L-dimension dynamics encoding. The MLP also has a hidden layer size hyperparameter, which is set to 128 for our purposes.

4.3. Intent Embedding and Encoder

Recall that the intent can take on one of 4 values. While we originally embedded the intent as a one-hot encoded vector, we also experimented with other embeddings for the intent. In particular, we map each of the 4 intents to the following natural language prompts:

1. "Go straight while avoiding obstacles"
2. "Go left while avoiding obstacles"
3. "Go right while avoiding obstacles"
4. "Stop while avoiding obstacles"

We then use a lightweight pre-trained language model - specifically MiniLM [16], which we chose for its relatively small model size but still high performance on natural language tasks. We then find the 384-dimension embeddings for each of these language prompts, and use this natural language embedding in place of the one-hot encoded intent. Finally, we use a 2-layer MLP to produce the final intent encoding in an E-dimensional vector. The MLP also has a hidden layer size hyperparameter, which is set to 64 for our purposes.

We find that while these intent embeddings were not learned from the data but rather came from the pretrained language model, they created a richer embedding for the intent compared to what a one-hot encoded vector could provide and thereby improved the performance of the model. We see that the language model's prior knowledge helps determine a intent embedding in this zero-shot manner that better encodes the complexities of each of the intents as a driving task better than a binary one-hot vector can.

4.4. Fusion Layer

At this point in the model architecture, we have extracted a D-dimension image feature vector, an L-dimension dynamics feature vector, and an E-dimension intent encoding. We finally need to fuse the outputs of these modules to predict the future trajectory's x,y position for the next 20 timesteps. To do so, we apply a final 2-layer MLP (with layer normalization) as a fusion layer that combines the encodings and outputs the final trajectory. The MLP also has a hidden layer size hyperparameter, which is set to 128 for our purposes.

4.5. Loss Function

We originally use Mean-Squared Error (MSE) as the loss function, as this is a standard loss used in trajectory prediction tasks [4]. The MSE loss between two trajectories, where each timestep in the trajectory contains an x,y coordinates, is defined in Eq. (1) below:

$$L_{\text{MSE}}(t, \hat{t}) = \frac{1}{N} \sum_{i=1}^{N} \left[\left(t_{i,x} - \hat{t}_{i,x}\right)^2 + \left(t_{i,y} - \hat{t}_{i,y}\right)^2 \right] \quad (1)$$

We also explored the use of an alternative loss function, the Smooth L1 Loss (also known as the Huber Loss, when δ is fixed to 1) [14].

$$L_{\text{SmoothL1}} = \frac{1}{N} \sum_{i=1}^{N} \Big[\text{smooth}_{\text{L1}}\left(t_{i,x} - \hat{t}_{i,x}\right) + \text{smooth}_{\text{L1}}\left(t_{i,y} - \hat{t}_{i,y}\right) \Big] \quad (2)$$

$$\text{smooth}_{\text{L1}}(r) = \begin{cases} 0.5r^2, & \text{if } |r| < 1 \\ |r| - 0.5, & \text{otherwise} \end{cases} \quad (3)$$

We can interpret the Smooth L1 loss to be a combination of L1 loss and the MSE loss (which is equivalent to L2 loss). This means that while MSE loss may heavily penalize outliers on the trajectory, the Smooth L1 loss will be less sensitive to these deviations and provide a smoother signal for more stable training and potentially finding better optima. This makes Smooth L1 loss particularly effective in when ground truth trajectories are noisy, which is especially the case in our scenario since we are dealing with a small dataset of abnormal driving situations.

We also incorporate early stopping during training, so we save the highest performing model weights seen throughout the epochs of training rather than saving the model weights determined after all epochs are complete.

4.6. Gradient Descent

To learn the weights, we mini-batch gradient descent [5]. Specifically, we use the AdamW optimizer [10], which combines the Adam optimizer with weight decay. We choose the Adam optimizer for its state-of-the-art performance on optimizing for weights of neural networks. We also include weight decay as it allows for better generalization - this particularly important for our task. Because of the low-data regime in which our model is being trained, the model is prone to overfitting and the controlled regularization of AdamW allows for generalizing the model to unseen data while also maintaining stability during training so the optimal model weights can be found through gradient descent. The learning rate, weight decay, and batch size were tuned as hyperparameters on the validation set.

4.7. Baseline

To determine a baseline to compare our model against, we want to build a simple model that ignores the visual input and simply uses $\mathbf{T}$ and the intent to predict the future trajectory. To do so, we will train an LSTM, which we know is well suited for the task of trajectory prediction. Specifically, the LSTM will take the past trajectory as input (with an appended one-hot encoded intent vector to each trajectory timestep). We will then pass final hidden state to a 2-layer MLP to output the predicted future trajectory.

5. Experiments

5.1. Setup and Hyperparameters

We select hyperparameters by tuning over the validation set. We chose to do a single fold given the constraints of training/evaluation time on our compute resources and size of the data (for reference, the data provided by Waymo was stored TFRecords, each of which are approximately 3 GB in size). The hyperparameters of each individual component are detailed in sections above. The high-level hyperparameters to note are the learning rate for gradient descent (chosen to be $\alpha = 1e-3$), the batch size (chosen to be 8), weight decay for AdamW optimization (chosen to be $\lambda = 1e-2$), and number of epochs (set to 200, but with early stopping).

After developing the model architecture, training the model on the train split, and tuning hyperparameters on the validation set, we finally run our model on the unseen test set. The training set has 1000 samples, the validation set has about 100 samples, and the test set has 263 samples.

5.2. Metrics

In addition to the loss value, the primary metric keep track of is the Average Displacement Error (ADE) metric [7], as recommended by Waymo to evaluate the performance of predicted trajectories. ADE is analagous the L2 error between trajectories. The loss (MSE or Smooth L1), also serves as a descriptive metric because it provides another measure for the difference between two trajectories. More specifically, for a given predicted trajectory $\hat{t}$ and the ground truth trajectory t over N timesteps, we can calculate the ADE as shown in Eq. (4) below. The total ADE across a batch of trajectories is therefore the sum of the ADE values across all trajectories in the batch.

$$\text{ADE}(t, \hat{t}) = \frac{1}{N} \sum_{i=1}^{N} \left[\left(t_{i,x} - \hat{t}_{i,x}\right)^2 + \left(t_{i,y} - \hat{t}_{i,y}\right)^2 \right] \quad (4)$$

6. Results and Discussion

The quantitative results for each iteration of the model is listed in Table 1. We summarize the results of the final model architecture as compared to the baseline Table 2 (note the LSTM baseline uses MSE loss while our model uses Smooth L1 loss). We notice that our model, which incorporates the vehicle's camera images among other changes, achieves better performance across the board. Also, we see that this performance generalizes well to both the validation and test sets. We can also observe that with more data, the performance of the model increases, which suggests that with more data and more compute resources, this architecture can provide even better performance on this task.

6.1. Qualitative Results

Figures 3 illustrates the qualitative performance of our trajectory prediction model across diverse driving scenarios. The top row shows a case where the model chooses to stop at a red light despite the "GO_STRAIGHT" intent, showcasing its ability to prioritize visual safety cues over semantic intent. The middle row highlights a complex maneuver, where the model successfully navigates through a road repair zone while avoiding cones, demonstrating its spatial awareness and planning competence. The bottom row illustrates a successful rightward trajectory prediction even at nighttime, reflecting the model's capability to interpret and perform well in adversarial conditions. These cases validate that the model effectively integrates visual context and intent to generate safe, situation-aware motion plans.

While the model exhibits strong performance across a range of driving scenarios, certain failure cases reveal its current limitations, as illustrated in Figure 4. In the top example, the model fails to predict a correct right-turn trajectory under challenging nighttime conditions. The scene is heavily affected by adverse weather—specifically rain, glare, and light dispersion from wet surfaces. Water droplets on the camera lens and poor illumination significantly degrade visual clarity, likely leading the model to misinterpret road boundaries and potential drivable space. In the bottom example, captured in a dense urban daytime environment, the model generates an inaccurate future trajectory. The scene is visually complex, with closely spaced vehicles and

Model	Description of Changes (trained over 250 sample subset)	Train Loss	Val Loss	ADE Train	ADE Val
0	LSTM Baseline (without Images, trained on 1k dataset)	1.235	4.198	35.600	49.521
1	ViT Backbone, Image Projection with Concatenation + Linear Layer, Past Dynamics MLP, intent encoder Linear layer, fusion MLP	0.222	12.178	6.083	62.779
2	Replace Intent Encoder with raw one hot values, Introduce Early Stopping	0.161	11.130	7.370	64.482
3	Replace One-Hot Intent with Intent Encoder Linear Layer + ReLU, Early stopping	0.159	11.153	6.764	60.691
4	Introduce Data Augmentation	0.900	7.989	17.566	55.374
5	Replace past dynamics MLP encoder with single LSTM	1.394	7.116	24.790	59.347
6	Replace MSELoss with SmoothL1 (Huber) Loss	0.337	0.980	23.001	43.903
7	Increased Batch Size from 8 to 32	0.386	1.106	13.834	48.428
8	Batch Size 16	0.380	1.086	16.691	52.448
9	Drop Batch Size down to 8, Introduce Dropout p=0.2	0.731	1.023	36.413	57.282
10	Removed dropout, Replace Intent Encoder with Introduce LLM embedding	0.407	1.016	17.122	44.125
11	Add a 2-Layer MLP Encoder after LLM Intent Embedding	0.353	0.982	13.182	41.102
12	Split the dynamics LSTM into a position LSTM and kinematics LSTM (concatenate LSTM outputs)	0.278	1.127	17.411	53.936
13	Introduce dynamics fusion MLP after two dynamics LSTMs	0.326	0.952	20.142	40.707
14	Introduce BERT to Replace the Final Fusion MLP	0.837	1.054	35.056	46.167
15	Replace BERT back to Fusion MLP, Unfreeze Last 2 Layers + Norm Layers of ViT	0.140	0.881	13.219	41.703
16	Introduce AdamW weight decay of 1e-2	0.146	0.847	10.493	39.197
17	Replace ViT with Swin Transformer	0.636	0.735	24.440	37.222
18	Introduce LayerNorm layer to Image Feature MLP and Final Fusion MLP	0.409	0.792	24.127	34.173
19	Train Over Full 1k dataset, Without Data Augmentation (for speedup)	0.611	0.613	31.333	26.688
20	Train Over Full 1k dataset, With Data Augmentation	0.606	0.663	25.937	29.237

Table 1: Evolution of Model Architecture (experiments above black line use MSE, while others use Smooth L1 loss)

	Train Loss	Val Loss	ADE-T	ADE-V
Baseline	1.235	4.198	35.600	49.521
Our Model	0.606	0.663	25.937	29.237
Final ADE of Model on Test Set: **26.96**				

Table 2: Summary of Results

pedestrians near the intended path. The model appears to struggle in such scenarios, likely misclassifying dynamic agents or occluded regions as static obstacles. This results in an overly cautious prediction that diverges significantly from the ground truth. We hypothesize that this limitation stems from the restricted visual context provided by using only the front three camera views. Incorporating all eight surround-view cameras could offer a more comprehensive understanding of the environment, enabling better reasoning in such cluttered scenes. Importantly, our model architecture is easily extensible to handle additional camera inputs, making it scalable to richer perception setups. These examples emphasize the importance of robust perception under both low-light and complex urban conditions. Improving resilience to visual degradation and better reasoning under occlusion remain key directions for future work.

Further analysis of the trajectories suggests that the model surprisingly also exhibits emergent behavior. As shown in Figure 5, the predicted trajectory fails to match the ground truth path. However, we can notice the reason for this difference is that the vehicle stops for the pedestrian rather than driving around them. This suggests that while the model may not exactly match human driving behavior, it still learns to plan its route safely in complex urban environments.

6.2. On the Utility of Model Components

We experimented with multiple options for each of the components, as demonstrated in Table 1. There were particular changes to the model architecture that realized the most performance improvement, and each of these high-performing changes together formed the final model. In particular, the following model changes provided the highest performance increases we observed through our experiments: (1) replacing MSE loss with Smooth L1 loss, (2) replacing the ViT with a Swin Transformer, (3) introducing data augmentation, and (4) embedding intent using natural language inputs.

We saw a performance increasing when changing the loss function from MSE loss to Smooth L1 loss. This is likely because the trajectory data in our small dataset was noisy, as expected since this dataset comprises of abnormal driving situa-

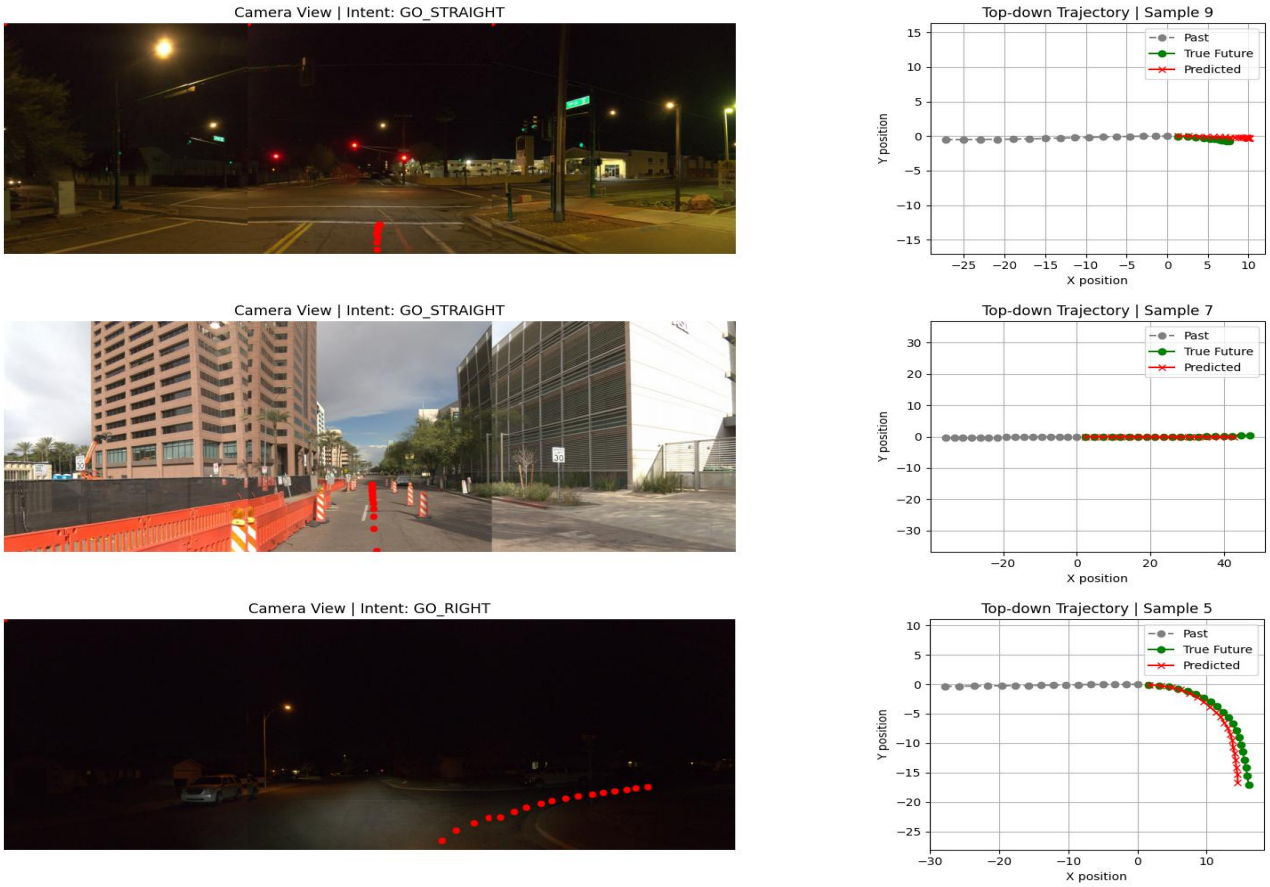

Figure 3: Success cases: **Top:** The model correctly constrains the vehicle to stop at a red light, despite the intent being to go straight. **Middle:** The predicted trajectory successfully avoids cones and navigates through a road repair zone. **Bottom:** Even under nighttime and adverse weather conditions, the model predicts a smooth right turn, demonstrating robustness to challenging environments.

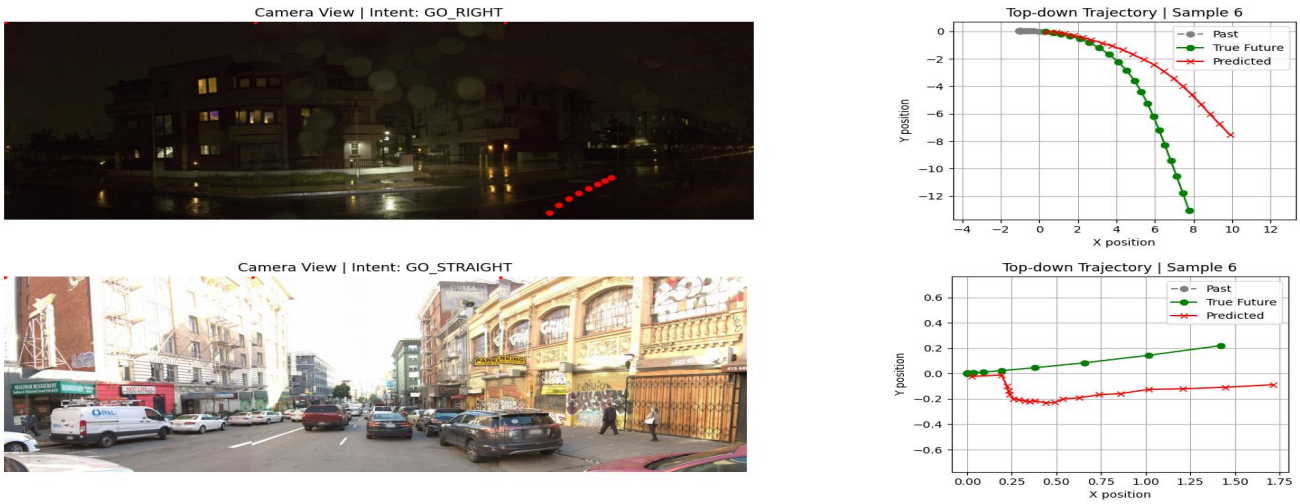

Figure 4: Failure cases: **Top:** Under low-light and rainy conditions, the model fails to predict a correct right turn, likely due to degraded visibility from glare, reflections, and water droplets on the lens. **Bottom:** In a dense urban environment, the model fails to predict the correct trajectory, potentially misinterpreting nearby pedestrians, moving and parked vehicles

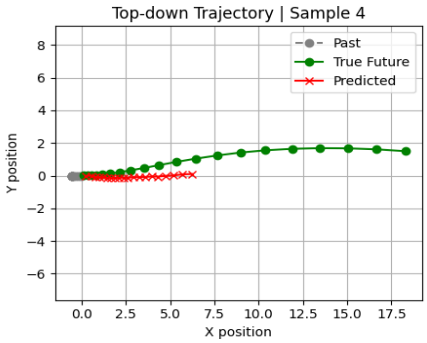

Figure 5: **Emergent Behavior:** Emergent deceleration behavior in response to a pedestrian crossing. Despite the "GO STRAIGHT" intent, the model predicts a significantly slowed trajectory without lateral deviation, implicitly learning to yield rather than reroute. This highlights the model's capability to perform nuanced motion planning under complex urban conditions

tions. As a result, MSE loss was heavily sensitive to these outlier trajectories compared to Smooth L1 loss, which was able to provide a better signal for gradient updates during optimization.

Replacing the ViT with the Swin transformer benefitted by both introducing smaller patch sizes without adding as many parameters to the model (which allowed for more fine-grained features within each patch) and takes advantage of the Swin transformer architecture that better allows it to extract low-level features from the image. This is particularly important in the driving scenario, where low-level features such as stop signs and line dividers can influence the predicted trajectories.

Data augmentation was particularly useful because of the limited training data we were using. So augmenting the data made the model more generalizable to unseen images and less overfit to our training set. Embedding intent using the natural language input rather than a one-hot encoded vector was the more surprising of performance increases seen across the experiments. We see that the richer embedding provided by the pre-trained LLM better conveyed the intent signal than a simple one-hot vector.

7. Conclusions & Future Work

In this work, we presented an end-to-end vision-based trajectory prediction framework that fuses multi-view camera input, past motion history, and natural language driving intent. Our architecture combines a Swin Transformer as a visual backbone, LSTM encoders for vehicle dynamics, and a MiniLM-based intent encoder, followed by a fusion network to predict future trajectories. Through extensive qualitative analysis, we demonstrated that the model can generate safe and context-aware trajectory predictions across a variety of driving scenarios. For example, it successfully identifies red lights and overrides the "go straight" intent to stop, maintains proper behaviour in the presence of road construction by driving within traffic cones, and performs reasonably well even under nighttime conditions.

Ablation studies highlight the most impactful design choices: switching from MSE to Smooth L1 loss improved robustness to noisy, long-tailed trajectory data; adopting the Swin Transformer allowed for better extraction of fine-grained road features (e.g., lane lines, stop signs); applying visual data augmentation helped generalization in a low-data regime; and replacing one-hot intent vectors with MiniLM-based natural language embeddings significantly improved semantic alignment and downstream prediction quality.

Despite strong performance, failure cases remain in low-visibility settings (e.g., rain, glare) and highly cluttered urban environments, where occlusions or poor lighting degrade visual signal quality and lead to conservative or incorrect predictions. We also observed early signs of learned collision avoidance behavior, as illustrated in Figure 5, although the model does not yet plan evasive trajectories around obstacles. Additionally, our current system lacks side and rear camera inputs, which contributed to certain failure cases during testing. These findings suggest that improving robustness to degraded visual input and reasoning under uncertainty remains an important direction.

One key limitation of the current architecture is the potential misalignment between independently pretrained components—specifically, the Swin Transformer for vision and MiniLM for language—which are frozen during training. As these modules are trained on separate modalities and objectives, their feature embeddings may not be optimally aligned for joint downstream prediction. In future work, we aim to replace these disjoint encoders with a unified vision-language model (VLM), such as Dolphins [11] (this seems very promising), which can jointly process image and text input within a shared embedding space.

Another promising direction is to explore diffusion-based trajectory predictors. Diffusion models offer stronger capabilities for conditional generation and can model uncertainty over multi-step predictions more effectively than deterministic regressors. Conditioning trajectory diffusion on fused visual, motion, and intent embeddings could yield more flexible and diverse predictions, particularly useful in ambiguous or multi-modal scenarios.

References

[1] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst, 2018. 2

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 3

[3] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation, 2020. 2

[4] A. Gopalakrishnan, A. Mali, D. Kifer, L. Giles, and A. G. Ororbia. A neural temporal model for human motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 4

[5] P. Jain, S. M. Kakade, R. Kidambi, P. Netrapalli, and A. Sidford. Parallelizing stochastic gradient descent for least squares regression: mini-batching, averaging, and model misspecification. *Journal of machine learning research*, 18(223):1–42, 2018. 5

[6] D. Kim, H. Shon, N. Kweon, S. Choi, C. Yang, and K. Huh. Driving style-based conditional variational autoencoder for prediction of ego vehicle trajectory. *IEEE Access*, 9:169348–169356, 2021. 2

[7] F. Liu, S. Duan, and W. Juan. A pedestrian trajectory prediction method based on improved lstm network. *IET Image Processing*, 18(2):379–387, 2024. 5

[8] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022, October 2021. 2

[9] W. LLC. Waymo open dataset: Vision-based end-to-end driving challenge (2025). https://waymo.com/open/challenges/, 2025. Accessed: June 4, 2025. 2

[10] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 5

[11] Y. Ma, Y. Cao, J. Sun, M. Pavone, and C. Xiao. Dolphins: Multimodal language model for driving, 2023. 8

[12] S. Moon, H. Woo, H. Park, H. Jung, R. Mahjourian, H. gun Chi, H. Lim, S. Kim, and J. Kim. Visiontrap: Vision-augmented trajectory prediction guided by textual descriptions, 2024. 2

[13] A. Seff, B. Cera, D. Chen, M. Ng, A. Zhou, N. Nayakanti, K. S. Refaat, R. Al-Rfou, and B. Sapp. Motionlm: Multi-agent motion forecasting as language modeling, 2023. 2

[14] I. Teeti, A. Thomas, M. Monga, S. Kumar, U. Singh, A. Bradley, B. Banerjee, and F. Cuzzolin. Astra: A scene-aware transformer-based model for trajectory prediction. *arXiv preprint arXiv:2501.09878*, 2025. 4

[15] B. Varadarajan, A. Hefny, A. Srivastava, K. S. Refaat, N. Nayakanti, A. Cornman, K. Chen, B. Douillard, C. P. Lam, D. Anguelov, and B. Sapp. Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction, 2021. 2

[16] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020. 4

[17] K. Yang, Z. Guo, G. Lin, H. Dong, Z. Huang, Y. Wu, D. Zuo, J. Peng, Z. Zhong, X. Wang, et al. Trajectory-llm: A language-based data generator for trajectory prediction in autonomous driving. In *The Thirteenth International Conference on Learning Representations*, 2025. 2

[18] J. Zhu, S. Qin, W. Wang, and D. Zhao. Probabilistic trajectory prediction for autonomous vehicles with attentive recurrent neural process, 2019. 2